UNIVERSITAT POLITÈCNICA DE CATALUNYA

Desenvolupament d'un sistema d'enviament de notificacions per proximitat

Amb la tecnologia Bluetooth Low Energy

14 d'octubre de 2016

Memòria del projecte que presenta Qiwei Ni
sota la direcció del Joan Martínez Domene
per assolir el grau d'Enginyer en Sistemes TIC.

Vull donar gràcies al Jordi,
al meu tutor Joan,
al departament DIPSE,
i a totes aquelles persones que m'han ajudat.

Índex

Abstract

This project creates a notification system by proximity using Bluetooth Low Energy (BLE). The solution consists of an electronic device with BLE that acts as a broadcaster of a point of interest, so that smartphones with BLE that are near the point of interest can detect it. The information of the point of interest will be delivered to an Android application, as well as it can be managed through a web administration panel.

This document consists of three parts. First, the development of an electronic device that acts as a locator. Second, building a locator device configuration interface. Third, creating an application for smartphones that detects the locator.

Resum

Aquest projecte consisteix en crear un sistema de notificacions per proximitat utilitzant la tecnologia Bluetooth Low Energy (BLE). La solució consisteix en un dispositiu electrònic amb BLE que fa de difusor d'un punt d'interès de manera que els telèfons mòbils amb BLE que es trobin aprop del punt d'interès el detectaran. La informació del punt d'interès es rebrà a una aplicació Android que s'ha desenvolupat i la gestió de la informació es farà a través d'un panell web d'administració.

Aquest document consta de tres parts. Primer, desenvolupament d'un dispositiu electrònic que fa de localitzador. Segon, construcció d'una interfície de configuració del dispositiu localitzador. Tercer, creació d'una aplicació pels telèfons mòbils que detecta el localitzador.

Part I.

Memòria

1. Introducció i estat de l'art

Quan passejem per una ciutat desconeguda o viatgem a un lloc nou, en algun moment es vol trobar un restaurant i ens agradaria saber quins menús ofereixen als restaurants més propers, sense haver d'apropar-nos a cada entrada per mirar les cartes. O bé, quan en algun teatre els hi queden unes últimes entrades hores abans de l'actuació, volen fer saber-ho a la gent. Aquest problema ja està resolt: es contracta a una persona perquè s'encarregui d'informar a la gent que circula aprop del restaurant o del teatre. Amb aquest treball s'ha volgut anar més enllà i desenvolupar una aplicació per mòbil que permeti rebre informació dels establiments que es troben aprop, de manera senzilla i ràpida tant per l'establiment com per l'usuari final.

1.1. Tecnologies de localització

Quan es vol saber la posició d'una persona, es parla de localització. Seguidament, es comenten les tecnologies (més comuns) de localització per mòbil.

GPS

El Sistema de Posicionament Global (GPS) [16z], és un sistema de navegació per satèl·lit que permet saber la situació geogràfica i l'hora de referència. El GPS funciona mitjançant una xarxa de satèl·lits que orbiten al voltant de la terra. Quan es desitja determinar la posició, l'aparell receptor de GPS detecta el senyal de com a mímim quatre satèl·lits de la xarxa, dels quals rep uns senyals indicant la posició i el rellotge de cadascun d'ells. Sobre la base d'aquests senyals, l'aparell sincronitza el rellotge del GPS i calcula el retard dels senyals, és a dir, la distància al satèl·lit. Per «triangulació» calcula la posició en què aquest es troba. La fiabilitat depèn del nombre de satèl·lits utilitzats, del grau de dispersió que tinguin aquests, de l'existència d'efectes atmosfèrics adversos que afectin a la velocitat de transmissió del senyal. Els GPS d'ús comú tenen un error de precisió de 15 m.

WPS

El sistema de posicionament Wi-Fi [16ac] s'utilitza pel posicionament en interiors però sovint fa de sistema complementari al GPS quan s'utilitza a l'exterior (per millorar la precisió). WPS ha sigut possible aprofitant el ràpid creixement a principis del segle XXI dels punts d'accés sense fil a les zones urbanes. La tècnica de localització més comuna i estesa utilitzada actualment es basa en mesurar la intensitat del senyal rebut. S'omple una base de dades amb la correlació de les dades d'ubicació GPS del dispositiu mòbil amb l'adraça MAC del punt d'accés Wi-Fi. Per tant, la presició depèn del nombre de punts d'accés que havia introduit en la base de dades.

Bluetooth Low Energy

Bluetooth Low Energy (BLE) [16ad], un dels 3 protocols de Bluetooth v4.0 [16ae], està dirigit a aplicacions de molt baixa potència (aparells que sovint només s'alimenten amb una pila de

botó). BLE no és compatible amb el protocol clàssic de Bluetooth, però utilitza les mateixes freqüències de ràdio, 2,4 GHz, i d'aquesta manera amb els xips que tenen BLE i Bluetooth clàssic poden compartir la mateixa antena. BLE té avantatges com: baix consum, mida petita, preu barat i compatible amb telèfons mòbils, tauletes i ordinadors.

Segons SIG (Bluetooth Special Interest Group), Bluetooth és s'utilitza pels dispositius propers, no indica la ubicació exacta com GPS.

1.2. Eines

Arduino Uno

Arduino Uno [16o] és una placa de circuit imprès basada en el microcontrolador ATmega328P. Té pins d'entrada i sortida digital i analògics, un cristall de 16 MHz, un connector USB, un connector d'alimentació, *header* ICSP i un botó de reset. Arduino Uno conté tot el necessari per donar suport al microcontrolador. Es pot programar a través d'encarregar programari a port USB per exemple.

2. Situació del treball

En aquests últims anys, la gent està rebent una allau d'informació a través de la televisió, l'ordinador, el telèfon intel·ligent... i les empreses intenten utilitzar tots aquests mitjans que tenen disponibles per comunicar-se amb l'usuari final per vendre els seus productes. Els locals de les ciutats també estan fent servir alguns mitjans per ajudar o millorar els seus negocis. Com per exemple, cada cop més, més locals de menjar es disposen eines webs per vendre els seus productes.

Amb aquest projecte es proposa un nou canal de comunicació que permet als usuaris rebre informació dels locals per on desplacin als seus telèfons intel·ligents. Les possibles utilitats poden ser:

- Els restaurants poden enviar menús del dia.
- Els teatres poden avisar que hi ha unes últimes entrades per vendre de l'espectacle que es farà en poques hores.
- Les galeries d'art poden donar avisos que falta pocs dies per acabar l'exposició d'un pintor famós.
- Les parades dels transports públics poden oferir informació sobre les incidencies, canvis d'horaris, etc a les persones que estan esperant.
  ...

2.1. Objectiu

L'objectiu d'aquest projecte consisteix en: desenvolupar un sistema que informa als possibles usuaris propers als locals o espais d'interès d'una ciutat, a través d'enviament de notificacions per proximitat.

2.2. Estudi prèvi i solució

Per aconseguir l'objectiu que s'ha definit, es busca una solució per resoldre.

Es qüestiona quines tecnologies es poden determinar per poder localitzar la persona. Realitzem les següents propostes:

- Gps. Només funciona a l'aire lliure i consumeix molt ràpid la bateria del mòbil [16y]. Utilitzar GPS té un cost car perquè és un canal de comunicació molt lent, necessita per comunicar-se amb tres o quatre satèl·lits de pròrroga a 50 bit/s. S'exigeix l'alimentació de l'antena durant qualsevol comunicació. El pitjor és que mentre el GPS està activat, el sistema no pot entrar en mode son. El cost de la bateria del GPS es nota molt durant

l'adquisició inicial del missatge de navegació del satèl·lit: estat del satèl·lit, efemèrides i almanac. L'adquisició de cada satèl·li triga de 12 s a 30s, però si es necessita l'almanac complet, es pot arribar fin a 12 min. Durant tot això , el telèfon és incapaç d'entrar en mode son. Tot i tenint l'ajuda de A-GPS (GPS assistit) que resol parcialment el problema, enviant el missatge de navegació al seu dispositiu mòbil a través de la seva xarxa de dades cel·lular o fins i tot Wi-Fi, GPS continua gastant molta bateria.

- *Wi-Fi.* Aprofitant els punts d'accés Wi-Fi per determinar la posició. Requereix que els locals tinguin un punt d'accés Wi-Fi.
- GPS *i Wi-Fi.* Determina la localització de l'usuari utilitzant l'antena de telèfonia mòbil conjuntament amb el receptor de Wi-Fi, tant en interiors com en exteriors. Conté els avantatges i inconvenients anteriors.
- *Bluetooth Low Energy.* Com el seu nom indica és de baix consum. Una de les moltes funcionalitats que ofereix és detectar i informar de la proximitat d'un dispositiu BLE a curta distància. En aquest cas, es necessita un dispositiu BLE que emeti un identificador de manera que quan un mòbil amb BLE s'apropi a l'emissor, rebi l'identificador i conegui l'existencia de l'altre.

Després d'estudiar les característiques de cada tecnologia s'ha triat Bluetooth Low Energy per les següents raons:

- Permet geolocalitzar amb molta precisió.
- És un dispositiu barat, consumeix poca electricitat i ocupa poc espai.
- El dispositiu que ha d'estar al local d'interès pot comunicar-se directament amb els dispositius del voltant, a diferència del GPS.
- Cal que l'usuari disposi de la tecnologia que volem utilitzar. La majoria de telèfons mòbils intel·ligents que una gran part de la població posseeix ja incorporen BLE, de manera que l'usuari no té necessitat de comprar cap altre dispositiu, simplement instal·lant una aplicació al seu mòbil ja pot utilitzar el sistema.
- L'aplicació que s'executarà al telèfon mòbil ha de fer un ús convervador dels recursos per evitar que disminueixi dràsticament l'autonomia del dispositiu mentre l'aplicació està activa.

Tot seguit es plateja quin hauria de ser el funcionament de tot el sistema:

- El funcionament estarà basat en un sistema centralitzat controlat per una **organització**.
- El local o lloc d'interès sol·licita a la **organització** el dispositiu BLE que s'ha dissenyat, que té com a funcionalitat exclusiva emetre un identificador. D'ara en endavant, aquest dispositiu l'anomenarem **emissor**.
- L'identificador que envia l'emissor el rebran els telèfons mòbils (**receptors**) que es trobin a pocs metres de l'emissor.
- Els receptors que tinguin activa l'aplicació rebran l'identificador que ha enviat l'emissor i es comunicaran amb un servei web (administrat per la **organització**) que retornarà la informació associada amb aquell emissor.

- Quan l'aplicació mòbil tingui la informació dels emissors, la mostrarà a l'usuari amb forma de notificació.

2.3. Estudi del mercat

Actualment, ja existeixen dispositius BLE que fan d'emisors de contingut, com el que s'ha utilitzat en aquest projecte. S'anomenen *beacon* i s'utilitzen per fer difusions del seu identificador als dispositius BLE que es troben a prop.

Els beacons estan basats en els protocols: iBeacon, Eddystone i AltBeacon, especificats per les empreses Apple, Google i Radius Networks, respectivament.

En el mercat existeixen diversos fabricants que ofereixen un dispositiu *beacon*. Cadascú implementa el dispositiu de diferents maneres, però fan la mateixa funció. Hi ha unes poques empreses que en comptes d'oferir el dispositiu físic, el que ofereixen és el sistema de gestió dels beacons a través d'un panell web i un SDK per ajudar a desenvulopar una aplicació pròpia [16u].

El que diferencia aquest projecte dels altres és un sistema complet i centralitzat. És complet perquè és un sol projecte que ja té tots els elements necessaris: l'emisor, el panell d'administració i l'aplicació mòbil. És centralitzat perquè només existeix una sola aplicació pels usuaris, d'aquesta manera els usuaris no han de baixar més d'una aplicació per rebre la informació dels diferents locals i els propietaris dels locals no hauran de desenvolupar la seva pròpia aplicació.

3. Estructura i implementació del sistema

Per implementar la solució descrita a l'apartat anterior, necessitem els següents components:

- `Emissor BLE`: la seva feina és transmetre un identificador.
- `Servidor web`: per servir la pàgina web del panell d'administració dels usuaris del local (que permet configurar les notificacions per cada dispositiu BLE) i per servir les peticions que fa l'aplicació mòbil.
- `Base de dades`: per guardar el registre dels propietaris dels emissors, detalls sobre els emissors BLE, les notificacions de cada local...
- `Aplicació mòbil`: processa el senyal del dispositiu BLE, fa la consulta al servidor web i presenta a l'usuari final la notificació.

La figura 3.1 descriu el procés que es duu a terme des del dispositiu BLE fins a l'aplicació mòbil. La part d'esquerra de la figura mostra el procés de recepció de les notificacions i la part de la dreta mostra les interaccions que hi ha entre la base de dades i el panell d'administració. Els passos ordenats cronològicament són els següents:

1) L'emissor difon un identificador únic.
2) L'aplicació mòbil el detecta i fa unes peticions per consultar al servidor web.
3) El servidor consulta a la base de dades sobre el identificador que ha rebut el receptor.
4) La base de dades retorna la informació donant un identificador.
5) El servidor la retorna al mòbil.

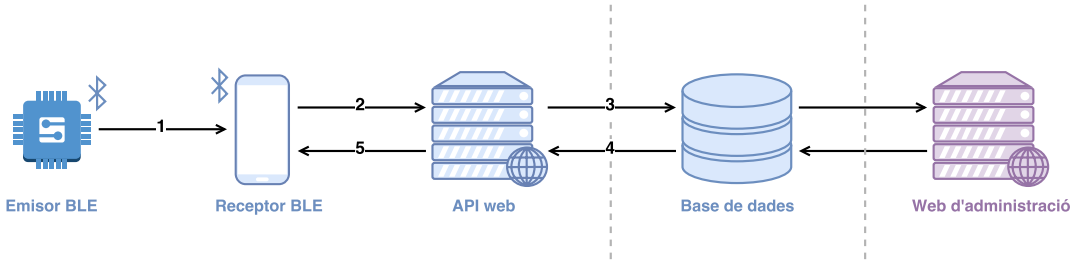

Figura 3.1.: Les connexions que hi ha entre diferents parts en conjunt.

3.1. Desenvolupament del dispositiu ble

S'utilitza un Radino nRF8001 [16v] per fer d'emissor, un mòdul que combina el microcontrolador ATmega32U4 i el circuit integrat nRF8001 [16x] (un xip de connectivitat BLE del fabricant Nordic Semiconductor), la figura 3.2 és un mòdul Radino. Les raons perquè s'ha escollit aquest mòdul són: primer, l'ATmega32U4 és molt semblant a l'ATmega328P (l'integrat que incorpora la placa Arduino Uno) i com que s'ha treballat molt amb aquest mòdul durant el grau, tenim una certa experiència; segon, Nordic Semiconductor proporciona una llibreria *ble-sdk-arduino* [16s] que facilita la comunicació entre l'integrat nRF8001 i les plaques Arduino (Arduino Uno, Mega, Leaonardo...).

Figura 3.2.: El mòdul Radino nRF8001

3.1.1. BLE

Els perfils de Bluetooth defineixen les funcions requerides i les característiques de cada capa en el sistema Bluetooth des de la capa més inferior a la superior i qualsevol altre protocol fora de l'especificació.

BLE té dos perfils genèrics, GAP i GATT. GAP (Generic Access Profile) [16a], la capa de control de BLE, gestiona connexions i radiodifusions en Bluetooth. GAP fa que el dispositiu sigui visible pels altres i determina com dos dispositius han d'interaccionar entre ells. Els dispositius BLE poden comunicar-se amb els altres de dues maneres: fent radiodifusions o establint connexions.

- `Radiodifusió`: enviament de dades unidireccionalment a qualsevol dispositiu que és capaç de recollir les dades transmeses.
- `Connexions`: intercanvi de dades entre dos dispositius, és a dir, bidireccional.

GATT (Generic Attribute Profile), la capa de dades de BLE, és una especificació general de la transmissió i recepció de trames curtes. GATT té les següents terminologies:

Usuari El dispositiu que inicialitza les ordres, les peticions GATT i accepta respostes. Per exemple, un ordinador o telèfons mòbils.

Servidor El dispositiu que rep les ordres, les peticions GATT i retorna respostes. Per exemple, un sensor de temperatura.

Característica (atribut) La dada que es transfereix entre usuaris i servidors. Per exemple, la temperatura actual.

Servei (atribut) La col·lecció de característiques, que es fan servir conjuntament per proporcionar una funcionalitat particular. Per exemple: servei de temperatura que conté característiques del valor de temperatura i la interval de temps de la mesura.

Descriptor (atribut) El descriptor conté informació addicional sobre les característiques. Per exemple, la unitat del valor de la temperatura. És opcional.

Identificadors s'assignen a serveis, característiques i descriptors i identificats per UUID. Bluetooth SIG té reservat un rang de UUID per als atributs estàndards, amb format de xxxxxxxx-0000-1000-8000-00805F9B34FB. Per comoditat, aquests identificadors es representen com a valors de 16 bits o 32 bits en lloc dels 128 bits per a un UUID complet.

En aquest projecte només necessitem que l'emissor BLE faci radiodifusions, és a dir, que es comporti com un *beacon*. Les radiodifusió envien paquets de difusió (advertising packet). Els paquets són enviats cada cert temps (interval de difusió). L'estructura d'un paquet estàndard de difusió està format per una trama de 31 bytes. Els paquets contenen estructures de dades de longitud variable. Cada estructura de dades està formada per 1 byte que indica la longitud, 1 byte que indica el tipus de dada de difusió (AD Type, Advertising Data Type) i els bytes restants són les dades [Tow+14].

S'ha escollit utilitzar el protocol Eddystone (basat en les especificacions de GATT) [16af], un perfil de beacon que Google va llençar el juny de 2015. Aquest perfil conté uns quants tipus de trama. La que encaixa amb el projecte es diu `Eddystone-UID`. Les trames de Eddystone-UID fan servir diferents tipus de dades:

- *Flags* indica quin mode de descobriment s'utilitza.
- UUID *de servei* identificador que informa de quin servei s'està oferint.
- *Dades de servei* dades addicionals associades al servei.

Segons el protocol de eddystone-UID [16t], la trama té el format representat a la taula 3.1.

trama 1			trama 2			trama 3		
mida	Flag	valor	mida	UUID servei	valor	mida	dada servei	valor
0x02	0x01	0x06	0x03	0x03	0xAAFE	0x17	0x16	0xAAFE...

Taula 3.1.: El format de trama Eddystone-UID

La següent taula 3.2 mostra detalladament la tercera trama:

UUID servei	valor				
	tipus	Tx power	10 bytes namespace	6 bytes instance	valors reservats
0xAAFE	0x00	...	...	...	0x0000

Taula 3.2.: El format de trama 3 Eddystone-UID

On:

- *Tx Power* és la potència de senyal rebuda a 0 m mesurats en dBm. El valor pot variar des de −100 dBm fins a 20 dBm a una resolució de 1 dBm. La manera de mesurar a 0 m és: mesurar a 1 m de distància i suma-li 41 dBm, ja que 41 dBm és la pèrdua del senyal que es produeix a 1 m.
- *10 bytes namespace* l'objectiu d'aquest camp és per tenir un ID únic dins de les implementacions de Eddystone i també es pot utilitzar per filtrar beacons.
- *6 bytes Instance ID* és ID de beacon, que pot estar definit en qualsevol manera adequada.

3.1.2. nRF8001

ACI (Application Controller Interface) és una interfície de comunicació sèrie bidireccional, entre l'nRF8001 i el microcontrolador. ACI utilitza SPI per intercanviar dades. El senyal CSN de SPI, que serveix per inicialitzar la transacció de SPI, és substituït pels dos senyals RDYN i REQN (active low handshake). L'nRF8001 notifica al controlador d'aplicació quan rep noves dades [Sem15].

- MISO: SPI Master In Slave Out
- MOSI: SPI Master Out Slave In
- SCK: SPI rellotge de dades serie
- REQN: controlador a nRF8001 handshake signal
- RDYN: nRF8001 a controlador handshake signal

El tràfic a través de ACI és bidireccional. En ACI hi ha dos tipus de dades d'intercanvi, ordres i esdeveniments:

Ordres (generades pel controlador) Poden ser ordres de sistema, que són per configurar l'nRF8001 i per controlar el mode de funcionament, o bé ordres de dades, per transferir dades amb un altre dispositiu a través de ràdio.

Esdeveniments (generats per l'nRF8001) Poden ser esdeveniments generats com a resposta d'un ordre, provocats per una condició predefinida (per exemple, esdeveniment de peer desconnectat) o perquè s'han rebut dades d'un altre dispositiu ràdio.

La figura 3.3 mostra el format d'un paquet. El *header* té una mida de 2 bytes: el primer byte és la mida del paquet restant (en bytes) i el segon byte és el codi d'operació. La resta de bytes són el *payload*, de mida variable.

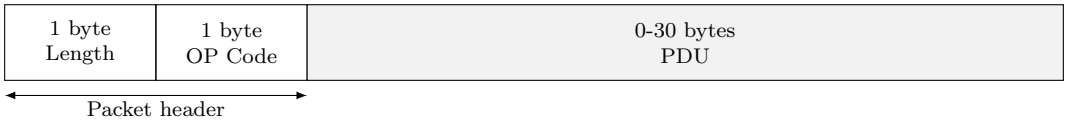

Figura 3.3.: L'estructura del paquet ACI

Els *Service pipes* de ACI serveixen per transmetre o rebre les característiques dels serveis. Els *broadcast pipes* permet difondre les dades de característiques que estan guardades a un servei

del servidor GATT de l'nRF8001. Per definir els serveis es pot fer utilitzant el programa `nRFgo Studio`, que proporciona el fabricant per configurar nRF8001. Un cop escollida la configuració, es transfereix al nRF8001 a través d'un microcontrolador.

En la figura 3.4, s'observa que està seleccionat el servei Broadcast, prèviament definit per anunciar. També estan seleccionats a l'apartat ACI *Broadcast* els camps *Local Services* i *Service data*.

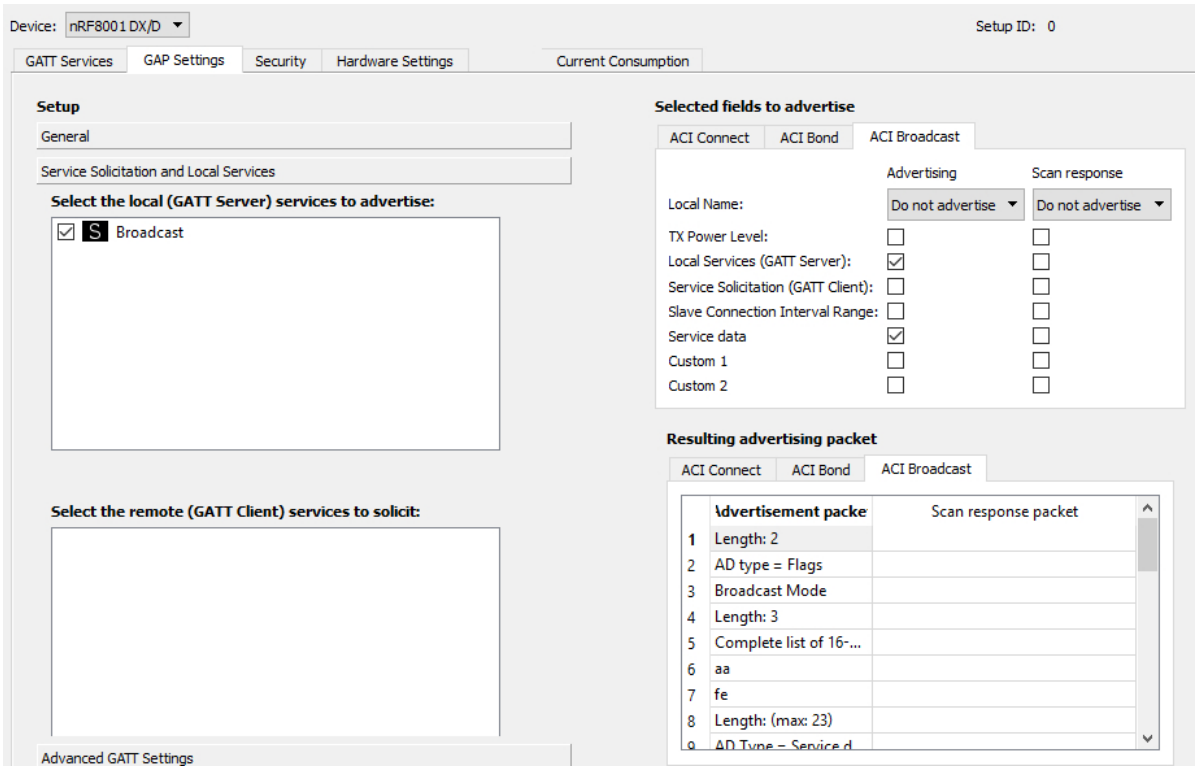

Figura 3.4.: Configuració de GAP en nRFgo studio

nRF8001 té 4 modes de funcionament: Sleep, Setup, Active i Test:

mode Sleep s'utilitza per parar totes les funcionalitats.

mode Setup s'utilitza per transferir les dades de configuració generades amb l'nRFgo Studio a la memòria de l'nRF8001. El procés de *setup* s'ha de completar perquè l'nRF8001 pugui ser utilitzat per enviar o rebre dades.

mode Active es fa servir per executar les ordres rebudes i per intercanviar dades. Per entrar aquest mode cal abans haver entrat al mode Setup. El mode Active té 3 submodes: mode Standby, quan està en repòs (l'nRF8001 entra en aquest mode després de completar el mode Setup); mode Advertising, quan està fent difusió o un dispositiu intenta connectar-se; mode Connected, quan ja està connectat amb un dispositiu.

mode Test s'utilitza per provar la connexió física ACI i la capa de RF PHY de l'nRF8001.

La funcionalitat de la llibreria *ble-sdk-arduino* està basada en comandes i esdeveniments de ACI, de manera que facilita molt en la implementació del programa. Aquesta llibreria proporciona diverses funcions de ACI, entre altres hi ha:

bool lib_aci_event_get(aci_state_t *aci_stat, hal_aci_evt_t * aci_evt) Aquesta funció obté l'esdeveniment de ACI des de la cua d'esdeveniments de ACI.

void lib_aci_init(aci_state_t *aci_stat, bool debug) Aquesta funció fa un reset a nRF8001.

uint8_t do_aci_setup(aci_state_t *aci_stat) Aquesta funció transfereix les configuracions generades des de nRFgo Studio a nRF8001 i retorna si ha anat bé o no. S'ha de cridar aquesta funció quan s'inicia o reinicia.

bool lib_aci_open_adv_pipe(const uint8_t pipe) Aquesta funció envia una ordre a la ràdio que col·loca el *pipe* `pipe` en les dades de servei de publicitat i retorna si ha sigut amb èxit. En el cas que `pipe` fos 1, vol dir que correspon a dades de servei de publicitat.

bool lib_aci_set_local_data(aci_state_t *aci_stat, uint8_t pipe, uint8_t *value, uint8_t size) Aquest funció actualitza el valor de característica o valor de descriptor de característica emmagatzemada localment al dispositiu.

bool lib_aci_broadcast(const uint16_t timeout, const uint16_t adv_interval) Aquesta funció comença radiodifusió en un interval de `adv_interval` mil·lisegons (poden ser de 100 ms a 10,24 s) amb un temps límit de `timeout` segons (en el cas que sigui infinit, el valor ha de ser 0).

En el inici de programa, es crida `lib_aci_init` per començar el mode Setup, cal cridar `do_aci_setup` per tranferir configuració a nRF8001 i completar mode setup per passar a mode Standby. En el mode Standby, cal obrir *broadcast pipe* utilitzant `lib_aci_open_adv_pipe` i després `lib_aci_set_local_data` per escriure dades de difusió, un cop està fet, crida `lib_aci_broadcast` per fer difusions periòdicament.

3.2. Serveis web

3.2.1. Descripció

Els serveis web que s'han desenvolupat en aquest treball tenen dues funcionalitats molt importants:

1) Oferir una eina senzilla, ràpida, còmode i múltiplataforma per escollir el contingut que emetran els punts d'interès.

2) Facilitar l'accés a aquesta informació a les aplicacions mòbils.

S'ofereixen a través d'Internet, utilitzant el protocol HTTP(S). Aquests serveis es poden biseccionar en quatre:

Panell d'administració per administrar la informació del punt d'interès.

API per Android perquè es pugui consultar la informació dels punt d'interès.

MariaDB base de dades relacional encarregada d'emmagatzemar tota la informació que es gestiona al panell d'administració.

Redis base de dades NoSQL que permet mantenir les sessions d'usuari obertes del panell d'administració.

3.2.2. Aplicacions web

Les aplicacions web són aquelles web que permeten a l'usuari interactuar com si es tractés d'una aplicació d'escriptori. El panell d'administració i l'API per Android són *aplicacions web*:

El panell d'administració permet:

- Registrar, modificar o esborrar beacon amb usuari per l'aministrador.
- Modificar la informació de beacon.
- Afegir, modificar o esborrar una notificació que està relacionat amb un beacon.

L'API per l'aplicació Android serveix per atendre peticions de l'aplicació:

- Retorna tots els tipus de beacon.
- Donat identificador de beacon i retorna informació de beacon i les seves notificacions.

Un criteri alhora de classifciar les aplicacions web (i altre programari) és segons la separació d'interessos entre una capa de presentació (*front-end*) i la capa d'accés a les dades (*back-end*) [16ag]. El panell d'administració està format per un *front-end* i *back-end* mentre que la API per l'aplicació Android només conté el *back-end* (el *front-end* és la pròpia aplicació Android).

S'ha utilitzat Flask [16ah] [16r] per desenvolupar les dues aplicacions web. Flask és un microframework de web que està escrit en Python i està basat en Werkzeug toolkit i Jinja2 template engine. Flask té com a objectiu matenir el nucli simple però és extensible. Existeixen moltes extensions que proporciona integració amb bases de dades, validació de formularis, administració de càrrega, diverses tecnologies d'autenticació, ...

Flask és tan simple que amb el següent codi ja es pot tenir una aplicació web que mostra ≪Hello World!≫:

```
from flask import Flask

# Creem l'aplicació
app = Flask(__name__)

# La funció 'hello' s'ha d'executar quan es demani el camí '/'
@app.route("/")
def hello():
    return "Hello World!"

@app.route("/bye")
def bye():
    return "Sayonara baby!"

# Arrenquem l'aplicació perquè comenci a esperar peticions
app.run()
```

Jinja2 és un dels motors de plantilles per Python més utilitzat. Està inspirat pel sistema de plantilla de Django, però està extès amb un llenguatge més expresiu que dóna facilitats als desenvolupadors i protegeix al desenvolupador d'errades que puguin provocar forats de segurertat com Cross-site scripting (XSS).

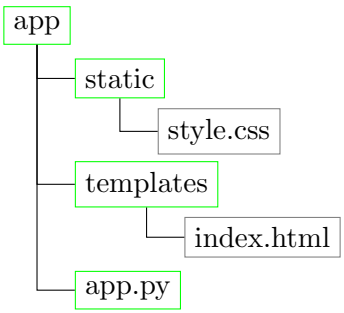

Figura 3.5.: L'estructura del app

La figura 3.5 representa l'estructura d'una aplicació Flask. Dins el directori `static` hi ha els fitxers que no canvien (que no són dinàmics), com els fitxers CSS i JavaScript. Dins el directori `templates` hi ha les platilles en format Jinja2 que utilitzarà Flask.

Les *routes* de Flask les podem classificar en dos tipus: les que retornen una vista (pàgina HTML) o les que simplement retornen dades en una notació específica.

En el cas del web de API només cal retornar informació necessaria i en format JSON (JavaScript Object Notation). JSON és un estàndard obert basat en text dissenyada per a intercanvi de dades llegible per humans. Deriva del llenguatge script JavaScript, per a representar estructures de dades simples i llistes associatives, anomenades objectes. El tipus MIME del JSON és `application/json`. El format JSON s'utilitza habitualment per serialitzar i transmetre dades estructurades en una petició. S'utilitza principalment per intercanviar dades entre un servidor i una aplicació web, sent una alternativa a l'XML. En web de API retorna el següent JSON quan es demana els tipus de Beacon:

```
{
  "r_code": 0,
  "data": {
    "tipus_beacon": [{
      "tipus_id": 1,
      "tipus": "bar"
    }, {
      "tipus_id": 2,
      "tipus": "restaurant"
    }]
  }
}
```

El servidor web [16ai] és un programari informàtic que s'encarreaga de rebre les ordres enviades amb el protocol d'HTTP, anomenades *peticions web*. Quan el servidor web rep una petició la delega a l'aplicació web. L'aplicació web s'executa, actua segons els paràmetres d'entrada, i retorna un resultat. Aquest resultat es retorna al servidor web i el servidor web l'envia a qui ha fet la petició web.

El servidor web serà la interfície d'accés per accedir i gestionar la informació, a través d'un panell d'administració web i una API web. La informació serà gestionada per una base de dades.

3.2.3. Base de dades

S'utilitzarà una base de dades relacional SQL, MariaDB [16aa], com el sistema de gestió de bases de dades. MariaDB és una branca de MySQL impulsada per la comunitat per tal de mantenir el seu estat lliure sota la GNU GPL.

Abans d'explicar l'estructura de la base de dades, es defineix:

- `usuari` propietari del local que té emissor BLE i que gestinen a través del web.
- `beacon` emissor BLE.
- `notificació` informació relacionat amb un emissor BLE.

Els `usuaris` tenen nom de usuari i contrasenya per identificar-se, cada usuari pot tenir un/uns `beacons`. Se li assiga un tipus concret als beacons depen del local on es disposa. El beacon s'identifica per uuid, conté l'informació d'adreça del local i número de telèfon. Cada un pot emetre en un temps vàlid, cap, una o més d'una `notificació` que podria ser un text pla, una imatge o ambdos. La base de dades té la següent estructura:

```
Usuaris(id, usuari, contrasenya)
Tipus_beacon(tipus_id, tipus)
Beacons(uuid, id_usuari, nom, tipus_id, adr, telf)
Notificacions(id, uuid, informacio, imatge, start, finish, creat, modificat)
```

L'figura 3.6 mostra l'esquema de base de dades.

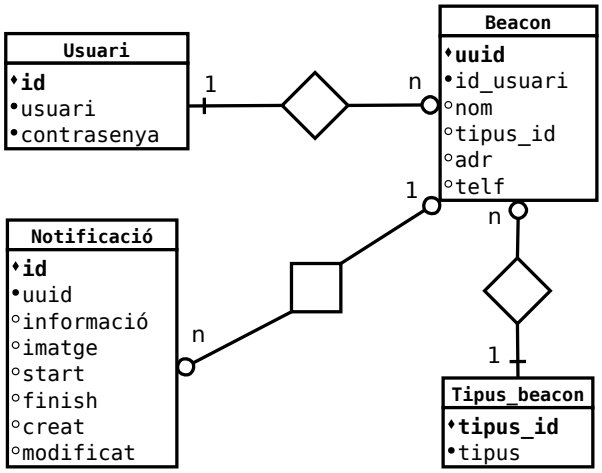

Figura 3.6.: L'esquema relacional

3.2.4. Infraestructura

A la figura 3.7 es pot observar que els quatre serveis estan aïllats entre sí. Aïllar els serveis en màquines diferentes és una bona idea perquè si un dels serveis web es veu compromès, serà més difícil comprometre els altres serveis. Separar els serveis web afavoreix més independència, flexibilitat i seguretat.

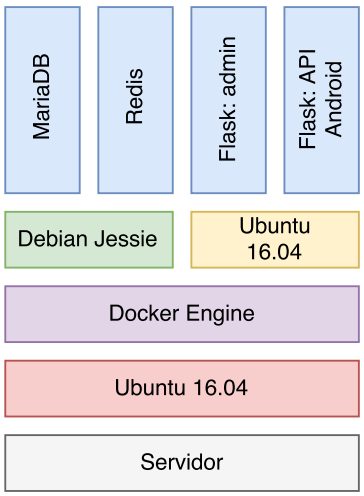

Figura 3.7.: L'estrucatura dels containers

Per aconseguir aquest aïllament existeixen diverses solucuions. A priori, un pot pensar en separar els serveis en quatre màquines (físiques) diferents, però implicaria haver de gastar més recursos en maquinària. Altres solucions permeten aconseguir l'aïllament dels serveis utilitzant només una màquina (física):

- *Emulació*: consisteix en un programari que emula una màquina física. Permet executar un sistema operatiu sencer. El rendiment és baix.
- *Virtualització*: la màquina física té un macanisme per compartir els recursos amb una o més màquina virtuals. L'hipervisor gestiona la virtualització de la màquina física. Permet executar un sistema operatiu sencer. S'obté un rendiment alt, comparable amb el d'una màquina nua (*bare metal*).
- *Contenidor*: es virtualitza a nivell de sistema operatiu, en comptes de virtualitzar a nivell de màquina. La capa entre la màquina física (*host*) i la màquina virtual (*guest*) és més fina. Permet obtenir un rendiment més alt que amb la virtualització. El sistema operatiu de la màquina *guest* ha de ser compatible amb el de la màquina *host*.

En aquest treball s'ha obtat per treballar amb contenidors, utilitzant l'eina Docker.

3.2.5. Construcció de les imatges

Docker [16p] proporciona la capacitat d'empaquetar i executar una aplicació en un entorn aïllat anomenat contenidor. L'aïllament i la seguretat li permeten executar molts contenidors de forma simultània en una màquina. A causa de la naturalesa lleugera dels contenidors, que s'executen sense la càrrega addicional d'un hipervisor, els recursos es poden aprofitar més.

Una característica destacable dels contenidors de Docker és que els canvis que es realitzen a una instància d'una imatge, són volàtils. Això sígnifica que si posem en marxa una instància de MariaDB, un cop es destrueixi, haurem perdut la base de dades. La manera de preservar fitxers en una instància consisteix en compartir una carpeta entre el `host` i el `guest`.

Per construïr imatges de contenidors de Docker es defineix una recepta amb totes les accions que s'han de dur a terme. El fitxer que conté la recepte s'anomena `Dockerfile`. D'aquesta

	nom	servei
1	mariadb	MariaDB 10.1
2	redis	Redis
3	flask-admin	Flask
4	flask-api-android	Flask4

Taula 3.3.: Serveis web

manera, podem construïr el contenidor simplement executant `docker build .` al directori on es troba el Dockerfile. Cada pas de la recepta (cada línia) que s'executi resultarà en una capa del container. Això significa que una imatge és la combinació d'una o més capes. Això permet aprofitar capes de diferents Dockerfile sempre que les anteriors siguin idèntiques.

En aquest projecte construïm una imatge pròpia basada en la imatge oficial d'Ubuntu 16.04.

Dockerfile

```
# Imatge base
FROM ubuntu:16.04

# Instal·lació bàsica de Python i dependències...
RUN apt-get update && apt-get install -y \
    python-pip python-dev build-essential libssl-dev libffi-dev

# Client de MySQL (compatible amb MariaDB)
RUN apt-get install -y libmysqlclient-dev

RUN pip install --upgrade pip \
    && pip install flask

# Application folder
ENV APP_DIR /app
WORKDIR ${APP_DIR}

# Script que s'executarà en arrencar la instància
COPY entrypoint.sh /entrypoint.sh

# expose web server port
# only http, for ssl use reverse proxy
EXPOSE 80

# exectute start up script (overwritten on docker-compose.yml)
ENTRYPOINT ["/entrypoint.sh", "<especificar-a-docker-compose>"]
```

3.2.6. Desplegament

Per realitzar el desplegament d'aquests serveis, ja sigui en un entorn de proves o d'explotació, s'ha optat per utilitzar *Docker Compose*. Docker Compose és una eina per definir i executar aplicacions de Docker multi-contenidor [16q].

Per fer-ho, es defineix en un fitxer anomenat `docker-compose.yml` la llista de serveis que requereix l'aplicació (veure la taula 3.3).

docker-compose.yml

```
version: '2'
services:
  flask-admin:
    build: .
    entrypoint:
     - /entrypoint.sh
     - admin
    ports:
     - "8080:80"
    volumes:
     - ./src:/app
    links:
     - redis
     - mariadb
    environment:
     - DB_HOST=mariadb
     - DB_USER=user
     - DB_PSWD=password
     - DB_NAME=database
     - REDIS_HOST=redis
     - SESSION_SECRET_KEY=\x1d\xfdv...\xb9^\xf4\x15

  flask-android-ws:
    build: .
    entrypoint:
     - /entrypoint.sh
     - ws
    ports:
     - "8081:80"
    volumes:
     - ./src:/app
    links:
     - mariadb
    environment:
     - DB_HOST=mariadb
     - DB_USER=user
     - DB_PSWD=password
     - DB_NAME=database
     - REDIS_HOST=
     - SESSION_SECRET_KEY=

  redis:
    image: redis:3
```

```
  mariadb:
    image: mariadb:10.1
    volumes:
      - ./db/scheme/:/docker-entrypoint-initdb.d
      - ./db/conf/:/etc/mysql/conf.d
      - ../docker-data/mariadb:/var/lib/mysql
    environment:
      - MYSQL_ROOT_PASSWORD=root_password
```

Tot seguit, només cal executar `docker compose up --build` per posar en marxa els serveis. Es podrà accedir al panell de control navegant a `http://localhost:8080/` i a la API de l'aplicació Android a `http://localhost:8081`.

Per qüestió de seguretat, s'ha creat un usuari per accés a la base de dades a través de la web. Aquest usuari té accés limitat a les taules de la base de dades i només pot accedir des de l'adreça d'IP 172.0.0.0/8 (són les adreces de la subxarxa dels contenidors de Docker):

```
CREATE USER 'usuari'@'172.%' IDENTIFIED BY 'contrasenya';
GRANT SELECT, INSERT, UPDATE ON TABLE beaconsDB.* TO usuari@'172.%';
GRANT UPDATE, DELETE ON TABLE beaconsDB.beacons TO usuari@'172.%';
GRANT UPDATE, DELETE ON TABLE beaconsDB.notificacions TO usuari@'172.%';
```

3.2.7. Entorn d'explotació

L'entorn d'explotació consisteix en un servidor virtual privat accessible des de `tfg-origin.qiwei.cat`. Aquest servidor de 1 core i 512 MB de RAM, amb Ubuntu 16.04 i Docker, executarà els serveis web i permetrà que siguin accessibles des d'internet.

El servidor ha estat llogat a `Digital Ocean`, una empresa que ofereix solucions *cloud* als desenvolupadors. Per comoditat, s'utilitzarà `CloudFlare` com a balancejador de càrrega, xarxa de distribució de contingut i per oferir SSL.

Per agilitzar el desplegament a l'entorn d'explotació, s'ha configurat el repositori `https://gitlab.jmr.cat/tfg-qiwei/api-web` per poder treballar seguint la metodologia de Continuos Delivery [16aj] (CD). Aquesta metodologia consisteix en aplicar els canvis en petits cicles sense comprometre la fiabilitat en fer el desplegament. D'aquesta manera, cada cop que s'envia un o més `commit`s al servidor de `git`, es publica automàticament la nova versió al servidor d'explotació.

El fitxer `.gitlab-ci.yml` conté els passos a seguir per realitzar la publicació:

```
prepare:
  stage: build
  script:
   - ln -s /docker-data ../docker-data
   - docker-compose build
  tags:
   - tfg-qiwei-prod-server
  only:
   - master
```

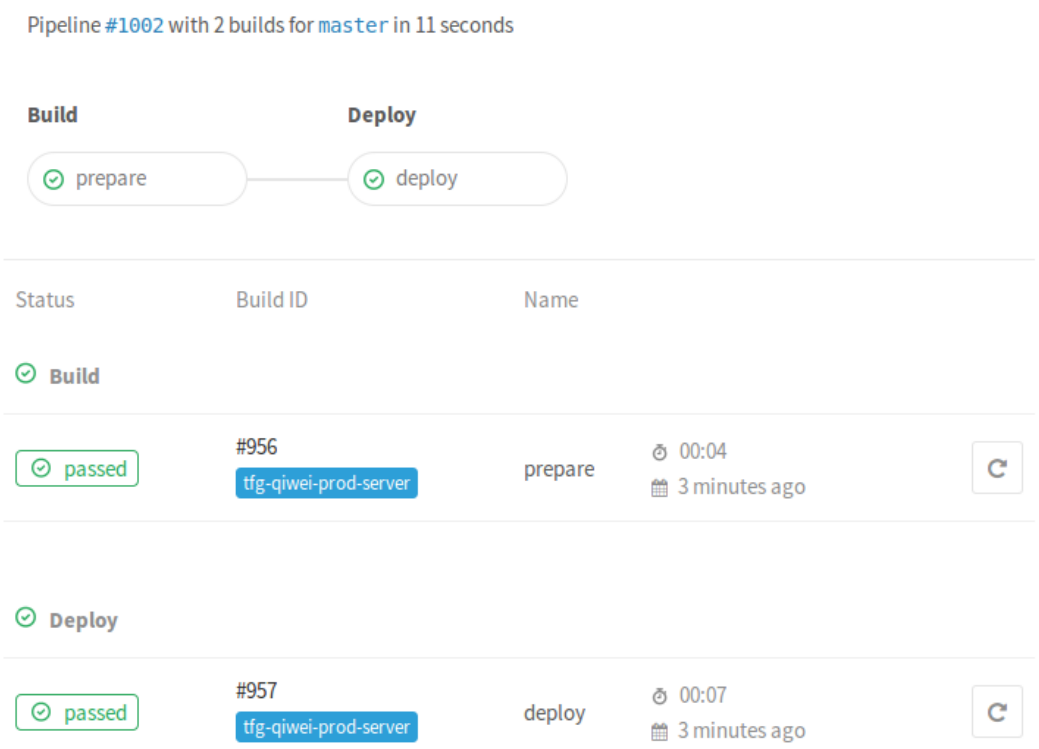

Figura 3.8.: GitLab CI

```
deploy:
  stage: deploy
  script:
   - docker-compose down
   - docker-compose up -d
  tags:
   - tfg-qiwei-prod-server
  only:
   - master

stages:
  - build
  - deploy
```

Després de realitzar un commit, observem que el desplegament ha anat bé en la figura 3.8.

3.3. L'aplicació Android

El funcionament principal d'aquesta aplicació consisteix en: escanejar per trobar els emissors BLE i consultar la informació al servidor web. El servidor retorna la informació i la mostra a l'aplicació. L'aplicació té les següents característiques:

- Permet a l'usuari seleccionar categories per filtrar les notificacions.
- Llista a l'usuari els emissors BLE pròxims i ordenats ascendentment segons la distància en que es troben.
- Notifica a l'usuari que s'ha detectat algun emissor BLE quan l'aplicació no està activa.

En els següents apartats, s'explicarà breument el sistema Android, els permisos d'Android, l'estructura de dades de l'aplicació, la implementació per fer peticions web al servidor, l'escaneig d'emissors i la integració en general.

3.3.1. Introducció d'Android

Android [16ab] és una plataforma formada per un conjunt de programari per a telèfons mòbils que inclou un sistema operatiu (construït sobre el *kernel* de Linux), programari intermediari i aplicacions. Android és la plataforma per a telèfons intel·ligents amb més vendes.

La primera versió comercial, Android 1.0, va ser llançada al setembre de 2008. Des d'aleshores, s'han publicat 14 versions. Aquest setembre es va publicar la versió 7.0, anomenada Nougat. Actualment, no tots els dispositius Android poden gaudir de la ultima versió, hi ha molta fragmentació. Les aplicacions acostumen a implementar-se donant suport les últimes versions, que tenen un percentatge d'usuaris superior al 5-10%. En general, és una bona pràctica donar suport al voltant del 90% dels dispositius [16b], i alhora, que l'aplicació sigui compartible amb l'ultima versió.

Les aplicacions Android s'executen sobre una màquina virtual de Java. Acostumen a estar escrites amb el llenguatge Java, conjuntament amb l'Android SDK. Android SDK inclou, entre moltes biblioteques, Jack, un compilador de Java que s'encarrega de compilar tots els móduls i els fitxers de recursos en un sol paquet, anomenat Android Application Package (APK). El fitxer `.apk` és un fitxer zip que conté totes les dependències de l'aplicació perquè un dispositiu Android pugui fer funcionar l'aplicació. Un cop l'APK està instal·lada en el dispositiu, cada aplicació està a la seva *sandbox*.

La figura 3.9 representa l'estructura de fitxers en un projecte de Android. Entre tots aquests directoris i fitxers cal destacar els següents:

AndroidManifest.xml proporciona informació essencial sobre l'aplicació i els seus components.

build.gradle definició de dependències i altres configuracions de construcció.

java directori que conté codi font Java.

res directori que conté els recursos de l'aplicació, com ara, fitxers dibuixables, fitxers de disseny, cadenes d'interfície d'usuari, etc.

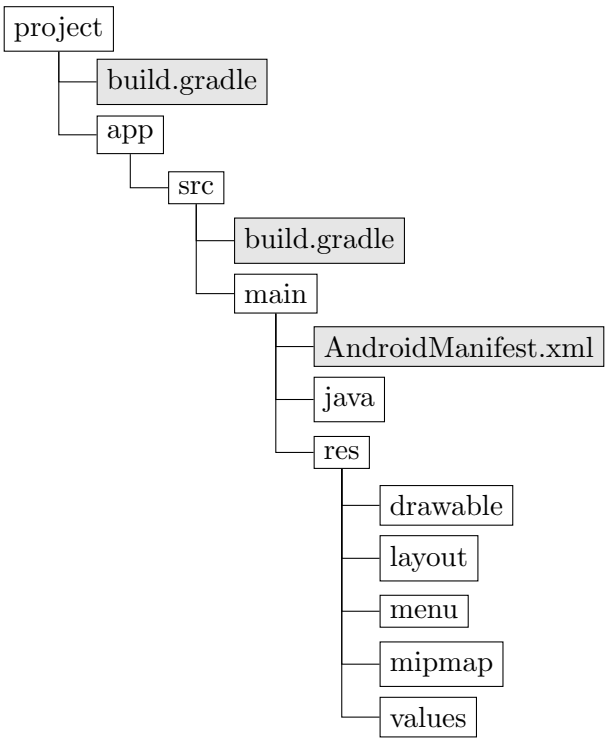

Figura 3.9.: L'estructura d'un projecte Android

Els components d'aplicació són elements essencials d'una aplicació Android [16c]. Hi ha 4 components fonamentals:

Activitats Element visual bàsic d'una aplicació que permet a l'usuari interactuar amb el dispositiu a través de la pantalla, per marcar un número de telèfon, enviar un correu electrònic, o veure un mapa. En general, totes les aplicacions tenen una activitat que s'especifica com l'activitat principal, que es presenta a l'usuari en iniciar l'aplicació.

Serveis Component que s'executa en segon pla per aconseguir una operació de llarga execució.

Proveïdors de contingut Capa d'accés a un conjunt de dades compartides entre aplicacions.

Receptors de difusions És un component que respon a difusions (*broadcasts*) de l'aplicació o de tot el sistema.

Per posar en marxa tres dels quatre components (activitats, serveis i receptors de difusions), s'utilitza l'enviament d'`intent`s. Un `intent` és una instancia de l'objecte `Intent` [16d], que conté un missatge per activar un component concret. En el cas d'activitats i serveis, l'`intent` defineix una acció a realitzar (utilitzant URIs). Per exemple, un intent podria transmetre una sol·licitud d'una activitat per mostrar una imatge o per obrir una pàgina web. En alguns casos, pot iniciar una activitat per rebre un resultat, en aquest cas, l'activitat també retorna el resultat d'un intent. En el cas de receptors de difusions, l'intent simplement defineix l'anunci que s'està emetent. Amb més detall seria:

- Per inicialitzar una activitat, s'ha de passar un Intent a la funció `startActivity()`, on Intent defineix l'activitat i les dades necessàries. Per rebre resultat d'una activitat quan s'acaba, cal implementar callbask de `onActivityResult()`.

- Per inicialitzar un servei, s'ha de passar un Intent a la funció `startService()`, on Intent defineix el servei i les dades necessàries.

- Per lliurar un missatge Broadcast, s'ha de passar un Intent a la funció `sendBroadcast()`.

En una aplicació, existeix diferentes activitats. Per gestionar-les, és important conèixer el cicle de vida: creació, arrencada, pausa, aturada i destrucció. Quan s'inicia una nova activitat, es pausa l'activitat anterior i el sistema conserva l'activitat en una pila. Quan s'inicia una nova activitat, que s'insereix a la pila de nou i es porta l'atenció a l'usuari. Quan l'usuari pressiona el botó enrere, s'extreu l'activitat de la pila (i es destrueix) i es reprèn l'activitat anterior. La figura 3.10 mostra cicle de vida d'una activitat, on els retangles representen les mètodes callback que es crida en l'execució [16e].

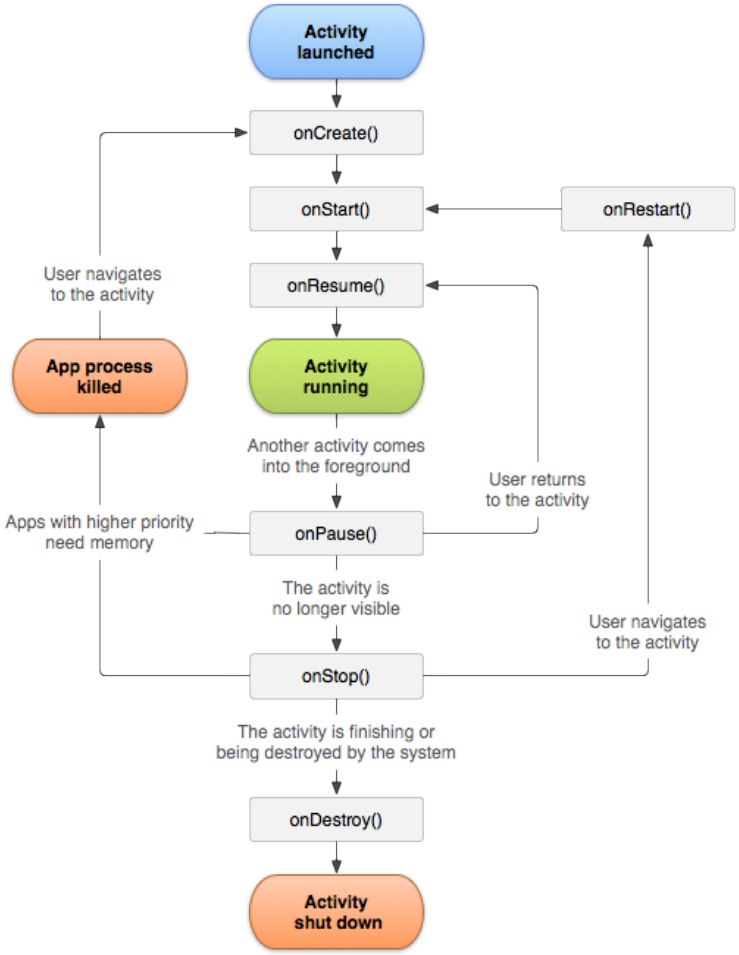

Figura 3.10.: Cicle de vida d'una activitat

3.3.2. Permisos

Android executa cada aplicació en un *sandbox*. Un *sandbox* és un mecanisme de seguretat que permet engabiar una aplicació per restringir l'accés a diversos recursos del sistema, com el disc,

la xarxa... Un cop l'aplicació està instal·lada en el dispositiu, cada una s'executa en el propi *sandbox*:

- Cada aplicació està instal·lada com a un usuari diferent.
- Per defecte, el sistema operatiu assigna un ID d'usuari únic per cada aplicació. El sistema configura els permisos dels fitxers de cada aplicació de manera que només hi pugui accedir la pròpia aplicació.
- Per defecte, cada aplicació s'executa en el seu propi procés i cada procés està a una instància de la maquina virtual de Java, per tant, les aplicacions s'executen aïllades entre sí.

Si l'aplicació vol utilitzar recursos i informació fora del seu *sandbox*, ha de sol·licitar permís explícitament [16f]. Hi ha dos tipus de permisos:

Permisos normals són aquells relacionats en accedir a dades, recursos que hi ha molt poc risc per a la privacitat de l'usuari. Per exemple, el permís per establir la zona horària o per canviar el fons de pantalla. El sistema concedeix aquests permisos automàticament si una aplicació els demana.

Permisos perillosos són aquells relacionats en accedir a la informació privada de l'usuari, que podrien afectar a les dades emmagatzemades o el funcionament d'altres aplicacions. Per exemple, permís de llegir els contactes de l'usuari. Aquests permisos els concedeix explícitament l'usuari si una aplicació els requereix.

A les versions d'Android anteriors a la 6.0 s'havien d'acceptar o rebutjar tots els permisos durant la instal·lació de l'aplicació. A partir de la versió 6.0, els permisos s'escullen en temps d'execució i individualment.

Aqueta aplicació necessita utilitzar Bluetooth per escanejar els emissors BLE i accés a internet per demanar informació sobre emissor BLE. En el fitxer de `manifest` es declaren els permisos:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.INTERNET" />
```

En la versió 6.0, per accedir identificadors dels dispositius externs a través d'exploracions properes Bluetooth, ha de tenir el permís de localització també [16g] [16h]:

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

3.3.3. Estructura de dades

S'han creat els següents models per cobrir les necessitats de l'aplicació:

BeaconType representa la categoria de l'emissor BLE, té un nom de categoria i el seu id.

BeconDevice representa la part física de l'emissor BLE, té atributs com, adreça MAC d'emissor, distancia en que es troba amb el mòbil, dades de paquet de difusió rebut ...

BeaconInfo representa la part abstracte de l'emissor BLE, té atributs com, nom de local, adreça de local, telefòn de local, categoria de local ...

BeaconNotification representa notificació definida pel local, té atributs com, nom de notificació, informació de notificació, imatges ...

Beacon representa un conjunt d'informació de l'emissor BLE, té atributs com, BeaconDevice, BeaconInfo i una llista de BeaconNotification.

Per no requerir constantment d'una connexió a internet per comunicar-se amb la API de l'aplicació Android va sorgir la necessitat de desar al disc el contingut d'alguns d'aquests models.

La primera opció que es va plantejar va ser **SQLite**, un sistema d'administració de base de dades relacional que a diferència d'altres sistemes d'administració de base de dades, no és un motor basat en client-servidor. SQL és un llenguatge conegut i SQLite és un producte madur, amb més de 16 anys. Per altra banda, fa pocs mesos apareix un nou producte: **Realm**. Realm és un motor de base de dades que es presenta com una alternativa a SQLite per aplicacions mòbils (iOS, Android i Windows Phone). Els principals avantatges són:

- Amb la definició del model com un objecte de Java, automàticament genera l'estructura de la base de dades.
- Es defineixen anotacions a la definició del model per indicar quin camp és la clau primària o un índex.
- Les consultes es realitzen encadenant condicions en forma de funcions amb paràmetres.
- La càrrega de dades del disc a la memòria és mandrosa (*lazy load*).
- Permet relacionar objectes, ofereix integritat i atomicitat (com la majoria de motors de base de dades).
- Facilita la sincronització de les dades entre fils de l'aplicació.

De manera que per crear un model, tan sols creem una classe de Java (que heredi de RealmObject), definim els atributs i alguna anotació:

```
public class BeaconType extends RealmObject {

    @PrimaryKey
    public long typeId;

    public String typeName;
    public boolean selected = true;
}
```

Ja podem començar a afegir objectes del tipus BeaconType a la base de dades. Per crear instàncies, modifcar els valors i fer cerques, es fa de la següent manera:

```
// Creem una instància de la base de dades (la primera vegada es construeix al disc)
Realm realm = Realm.getDefaultInstance();

// Comencem una transacció
```

```
realm.beginTransaction();

// Creem dos BeaconType (indiquem el tipus de l'objecte i la clau primària)
BeaconType tipus1 = realm.createObject(BeaconType.class, 1);
BeaconType tipus2 = realm.createObject(BeaconType.class, 2);

// Assignem alguns valors
tipus1.typeName = "Restaurants";
tipus2.typeName = "Museus";

// Apliquem els canvis (atòmicament)
realm.commitTransaction();

// Cerquem un BeaconType donat un 'typeId'
BeaconType tipus1_cerca = realm.where(BeaconType.class)
                               .equal("typeId", 1)
                               .findOne();

// Verifiquem la cerca
if (tipus1.typeId == tipus1_cerca.typeId)
    Log.i("I", "Acabem de fer una cerca!"); // => Acabem de fer una cerca!

// Tanquem la instància de realm
realm.close();
```

3.3.4. Serveis web

L'aplicació necessita saber la llista de categories dels locals i també cal saber la informació dels emissors que es troben. Tota aquesta informació no està guardada en el dispositiu, ja que els administradors dels emissors la poden actualitzar qualsevol moment. Per tant, es necessita consultar al servidor web d'API cada cop.

Fer peticions és una operació que pot implicar retards impredictibles i això pot causar una mala experiència de l'usuari. Per evitar això, cal realitzar-ho en un fil separat de la interfície d'usuari. Android ofereix la classe `AsyncTask` que permet tenir operacions en segon pla i retorna el resultat en el fil de la interfície d'usuari sense haver de manipular els fils [16i].

`Volley` [16m] és una llibreria d'HTTP que permet realitzar peticions HTTP en Android de manera més fàcil i més ràpida. `Volley` ofereix: la programació automàtica de les peticions, múltiples connexions de xarxa simultànies, emmagatzematge de les respostes en cache, suport a la petició de priorització, API de cancel·lació de peticions, personalització de les peticions, etc. `Volley` suporta tipus de peticions comuns, com ara, cadenes, imatges, JSON.

S'ha implementat unes classes que hereten de `Volley.Request<JSONObject>` per fer les peticions:

- `GetBeaconTypes` permet fer consultes de les categories de locals a través del mètode GET a `ws.tfg.qiwei.cat/beacon_types`
- `GetBeaons` permet fer consultes de la informació dels emissors trobats a través del mètode POST amb un JSON que conté els identificadors dels emissors a `ws.tfg.qiwei.cat/notifications`

3.3.5. Ble de Android

La classe `BluetoothLeScanner` ofereix mètodes per fer operacions d'escaneig dels dispositius BLE [16j]:

startScan Aquest mètode comença a escanejar. Els resultats d'escaneig són lliurats a través d'un callback.

stopScan Aquest mètode para l'escaneig.

La classe de `IntentService` proporciona una estructura senzilla per a l'execució d'una operació en un segon pla. Això li permet gestionar les operacions de llarga durada sense afectar la interfície d'usuari. A més, un `IntentService` no es veu afectada per la majoria dels esdeveniments del cicle de vida d'interfície d'usuari. `IntentService` té el mètode `onHandleIntent` que s'invoca quan sol·licita per processar [16n].

Això és el que necessita l'aplicació per estar contínuament escanejant i notificar a l'usuari quan troba algun emissor amb notificacions. Per obtenir els resultats del servei s'utilitza `LocalBroadcastManager`, és una classe que ofereix enviament d'intents a tots els components d'aplicació que està registrat a rebre. En el cas de l'aplicació, cal enviar l'intent quan es troba un emissor amb notificacions i els components que estan registrats quan reben l'intent, fan el que toca, per exemple, l'activitat principal refresca la interfície quan rep l'intent.

3.3.6. Interfície

La part d'interfície d'usuari de l'aplicació està dissenyada seguint les pautes de Material Design [16w], una guia completa per disseny de les aplicacions Android.

La interfície d'usuari de l'aplicació és tot el que l'usuari pot veure i interactuar. Android ofereix una varietat de components preconstruïts. Un *layout* [16k] defineix l'estructura visual d'una interfície d'usuari, com ara les interfícies per les activitats. Es pot declarar un *layout* de dues maneres:

- Declara elements d'interfície en un fitxer XML que es guarda en directori `res/layout/`.
- Instancia elements *layout* en temps d'execució.

L'avantatge de declarar interfície en XML és separar clarament la part de codi i la part visual, es pot modificar fitxer XML sense canviar codi font. Un exemple de *layout* podria ser el següent, on els elements tenen diferents atributs, per exemple, `id` serveix per identificar els elements i es pot instanciar utilitzant la funció `findViewById()` passant la referència de recurs en format `R.id.id_name`; `layout_width` i `layout_height` són per definir la mida de l'element:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello, I am a TextView" />
```

```
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello, I am a Button" />
</LinearLayout>
```

Es carrega a una activitat a través de la funció `setContentView()` passant la referència de recurs *layout* en format: `R.layout.layout_file_name`:

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

3.3.7. Integració

L'aplicació està feta amb IDE oficial de desenvolupament d'aplicacions Android, Android Studio [16l]. Aquest IDE proporciona les eines més ràpides per a la creació d'aplicacions en cada tipus de dispositiu Android, té editor de text, editor de disseny, eines de depuració, eines de rendiment, un sistema de construcció flexible i un sistema de generació/desenvolupament instant ...

L'aplicació inicialitza `IntentService` per escanejar els emissors BLE. Quan troba un emissor, es comprova amb la llista de Beacons, en el cas que no existeixi es fa una consulta al servidor web de API amb l'identificador, si existeix la informació d'emissor, es guarda en la llista de Beacons i es fa un *broadcast* d'actualització de la llista de Beacons. En el cas que ja existeixi el Beacon, s'actualitza la distància del Beacon i també es fa un *broadcast*. En les activitats que estan registrades d'aquest *broadcast*, es refresca la interfície. Quan l'usuari selecciona o desselecciona algun BeaconType, també refresca la interfície per mostrar només els Beacons que tenen BeaconType seleccionat.

4. Producte final

En aquest capítol s'ensenya el resultat de les tres parts del producte final, que són l'emissor BLE, la web del panell d'administració i l'aplicació Android.

Emissor ble

Per protegir del món exterior i per comoditat, disposa una caixa de plàstic per ficar-hi el Radino. En fase de desenvolupament, el Radino s'alimentava a través del port USB d'ordinador, però no és una manera pràctica pels locals. S'ha afegit una bateria a la caixa perquè el Radino pugui estar alimentat.

Panell d'administració

La web disposa de registre i d'inici de sessió pels propietaris dels emissors BLE (veu la figura 4.1). Si entra a la sessió d'administrador de web, hi ha un apartat de menú que posa administració (veu la figura 4.2). En l'apartat d'administració hi ha una taula que mostra l'identificador de l'emissor i el seu propietari (veu la figira 4.3), es pot editar clicant l'enllaç a la part dreta de la taula i registrar, clicant el botó groc, llavors entra la pàgina de registre d'un emissor (veu la figura 4.3). Si s'entra amb un usuari normal, el menú només mostra apartats Beacons i Notificacions (veu la figura 4.5). A l'apartat Beacons mostra la taula de Beacons que té l'usuari (veu la figura 4.6) i l'enllaç d'editar per un Beacon concret (veu la figura 4.7). A l'apartat Notificacions mostra una taula de notificacions que té afegit l'usuari (veu la figura 4.8).

L'aplicació Android

L'aplicació es troba en el distribuïdor d'aplicacions Google Play perquè qualsevol persona que tingui un dispositiu Android el pugui baixar. En la pantalla principal de l'aplicació es mostren les notificacions que s'han trobat (veu la figura 4.9), a la part de dalt de l'esquerra hi ha una icona del menú (veu la figura 4.10), quan es clica, es desplega un menú on es poden seleccionar les diferents categories d'informació. En aquest menú també es pot escollir per veure en les notificacions agrupades pels establiments (veu la figura 4.11). Cada element que hi ha a la pantalla, es pot veure amb més detall clicant-li (veu la figura 4.12).

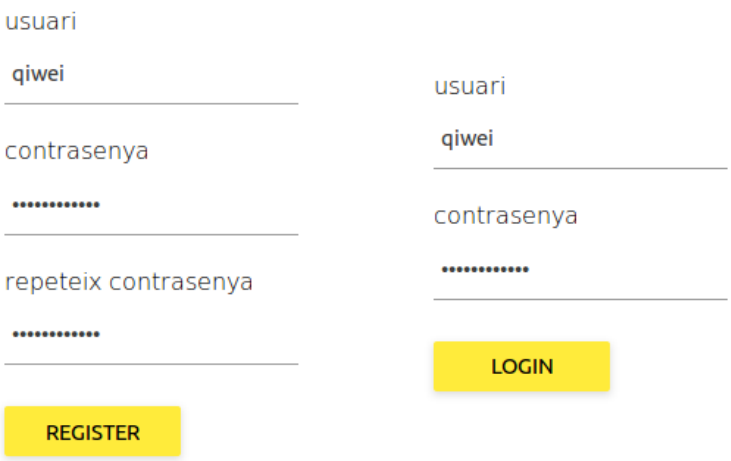

Figura 4.1.: Registre i Login de panell d'administració

Figura 4.2.: Menu de la sessió administrador

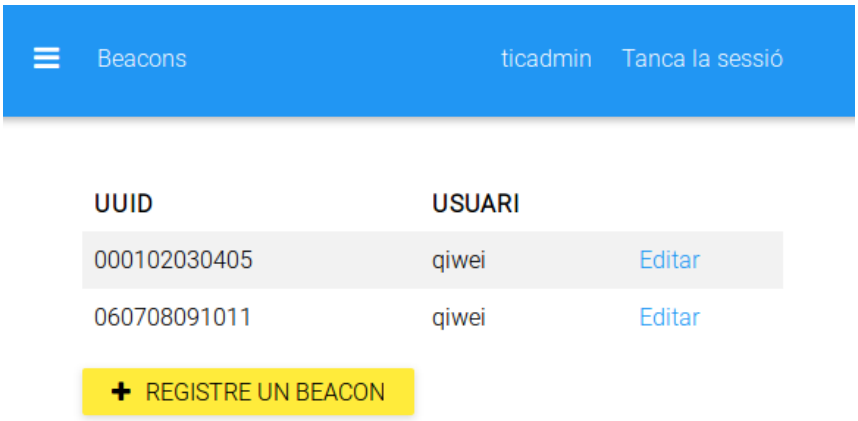

Figura 4.3.: Pàgina principal de l'apartat administració

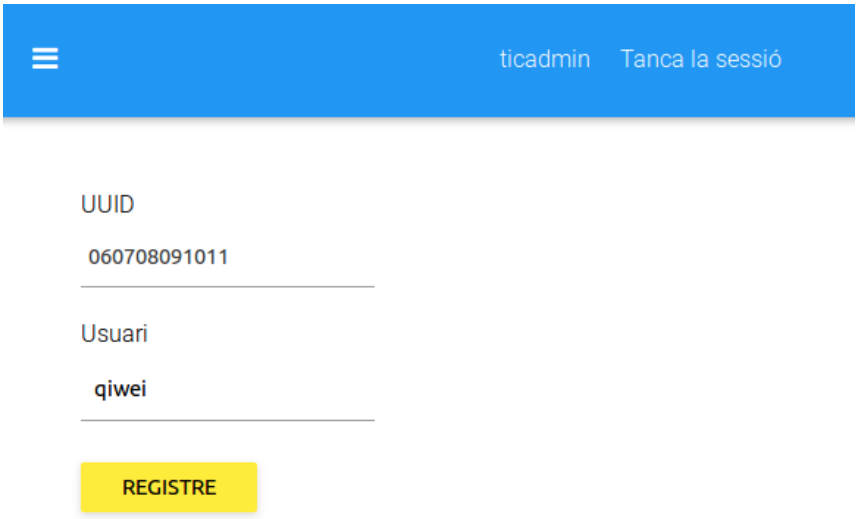

Figura 4.4.: Pàgina de registre d'un emissor

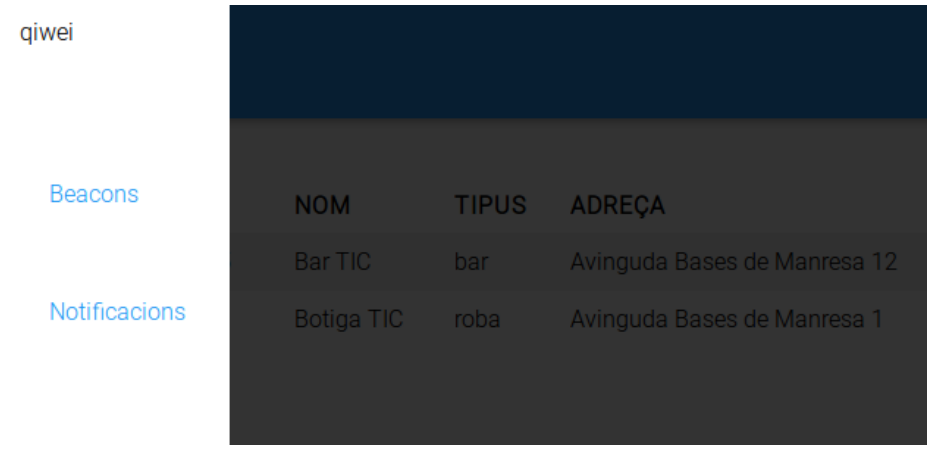

Figura 4.5.: Menú de la sessió d'usuari

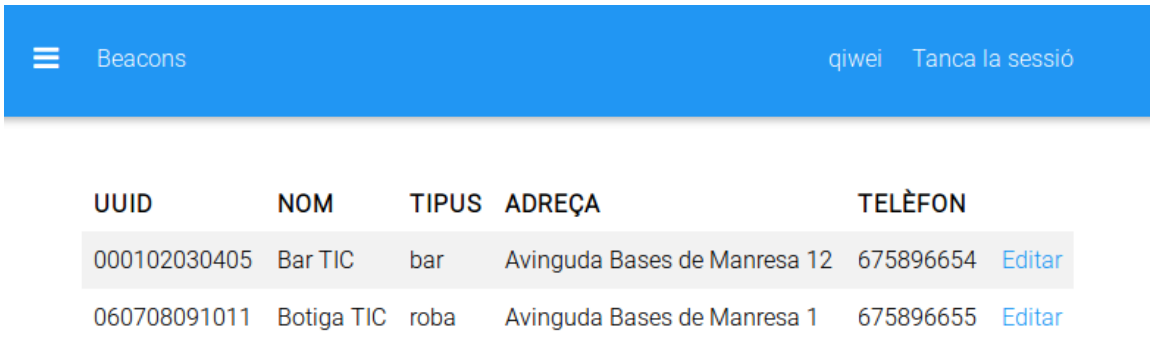

Figura 4.6.: Pàgina principal de l'apartat Beacons

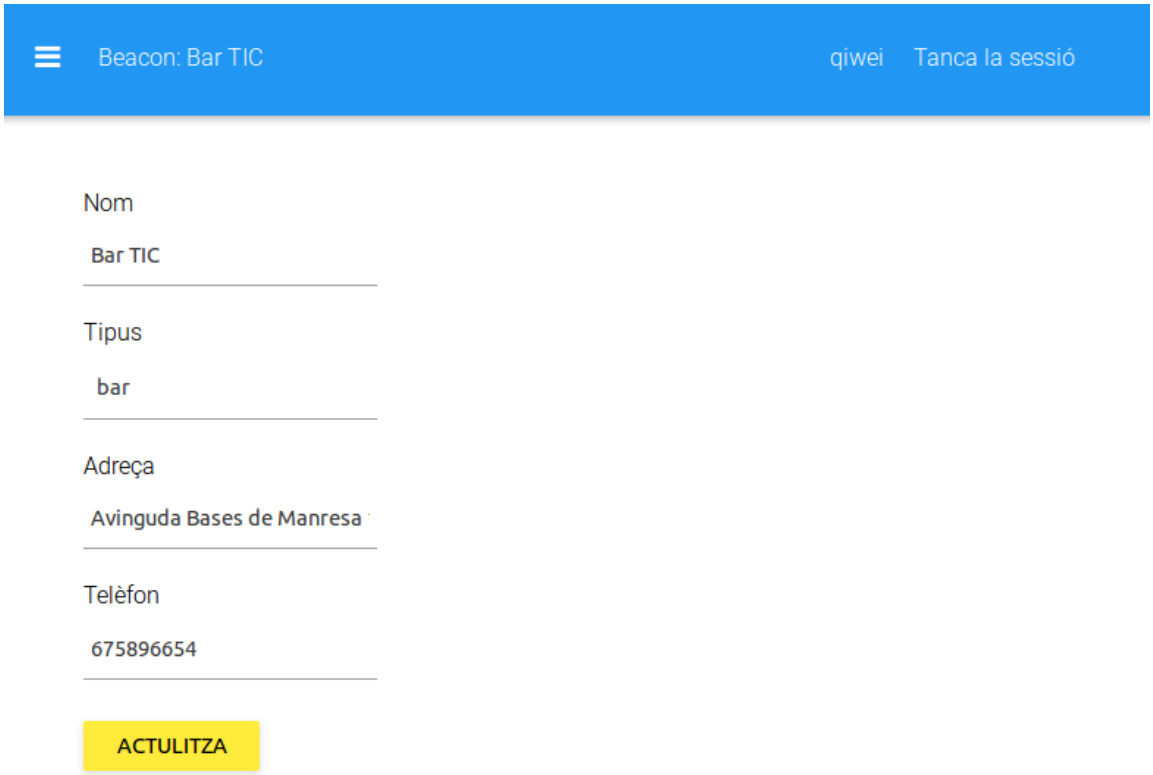

Figura 4.7.: Pàgina per editar un Beacon

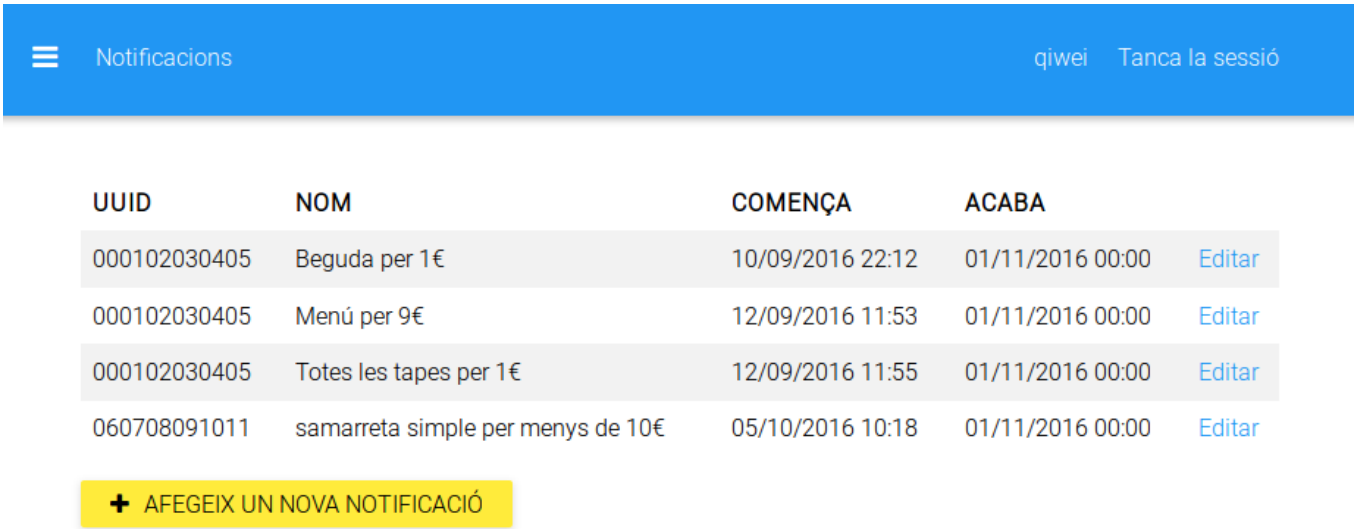

Figura 4.8.: Pàgina principal de Notificacions

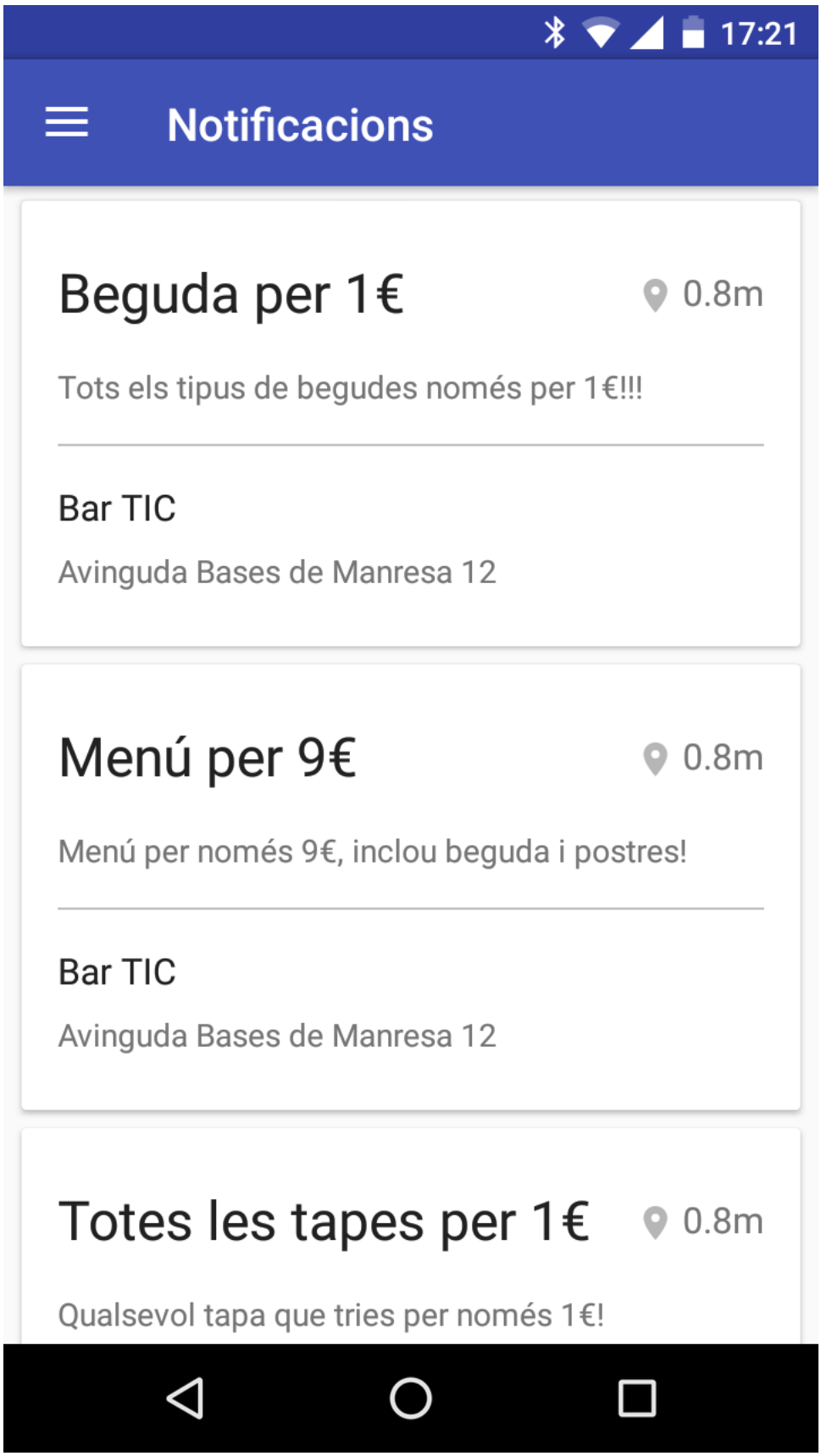

Figura 4.9.: La informació rebuda de l'aplicació veure en format de notificacions

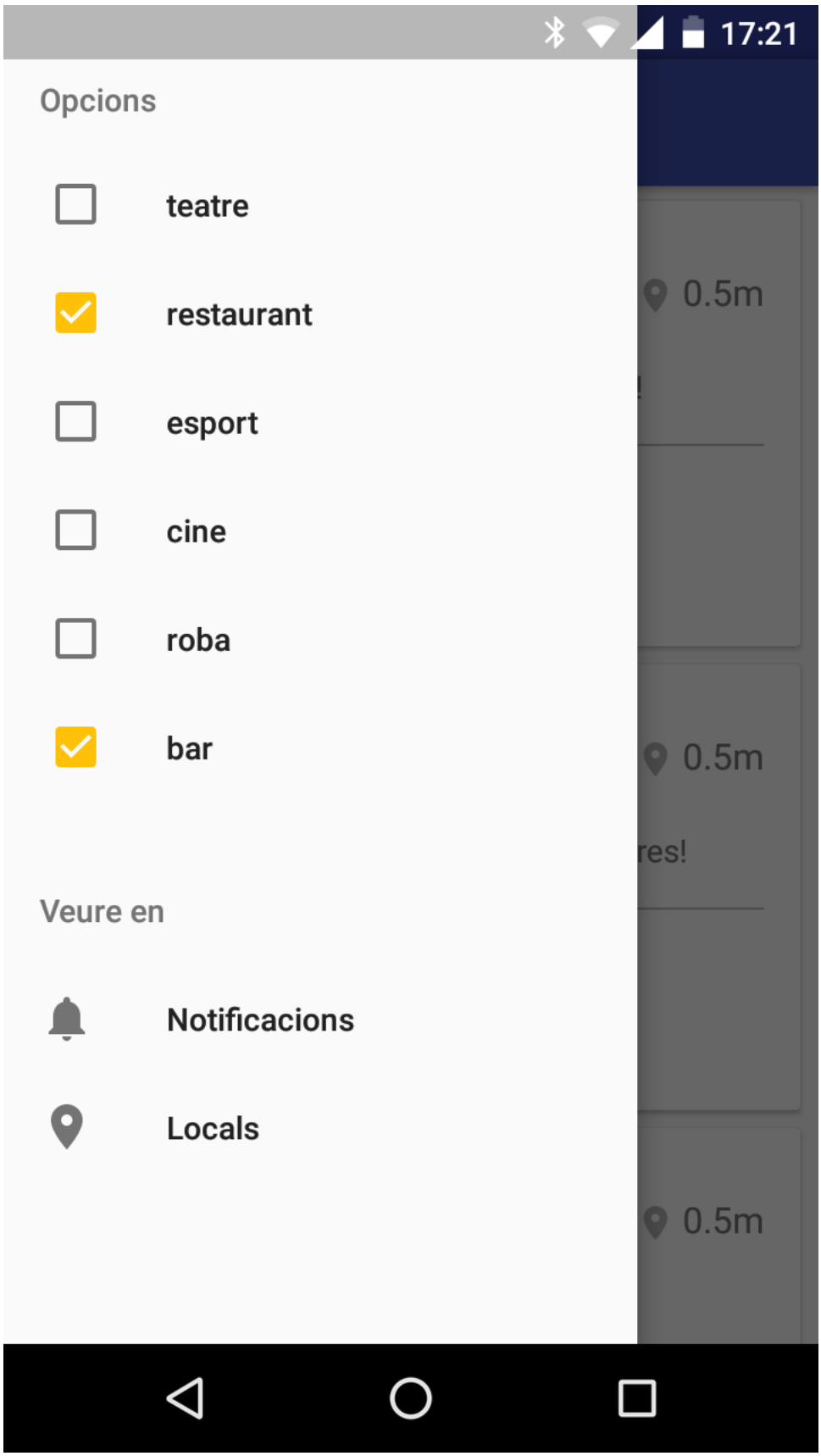

Figura 4.10.: El menú de l'aplicació

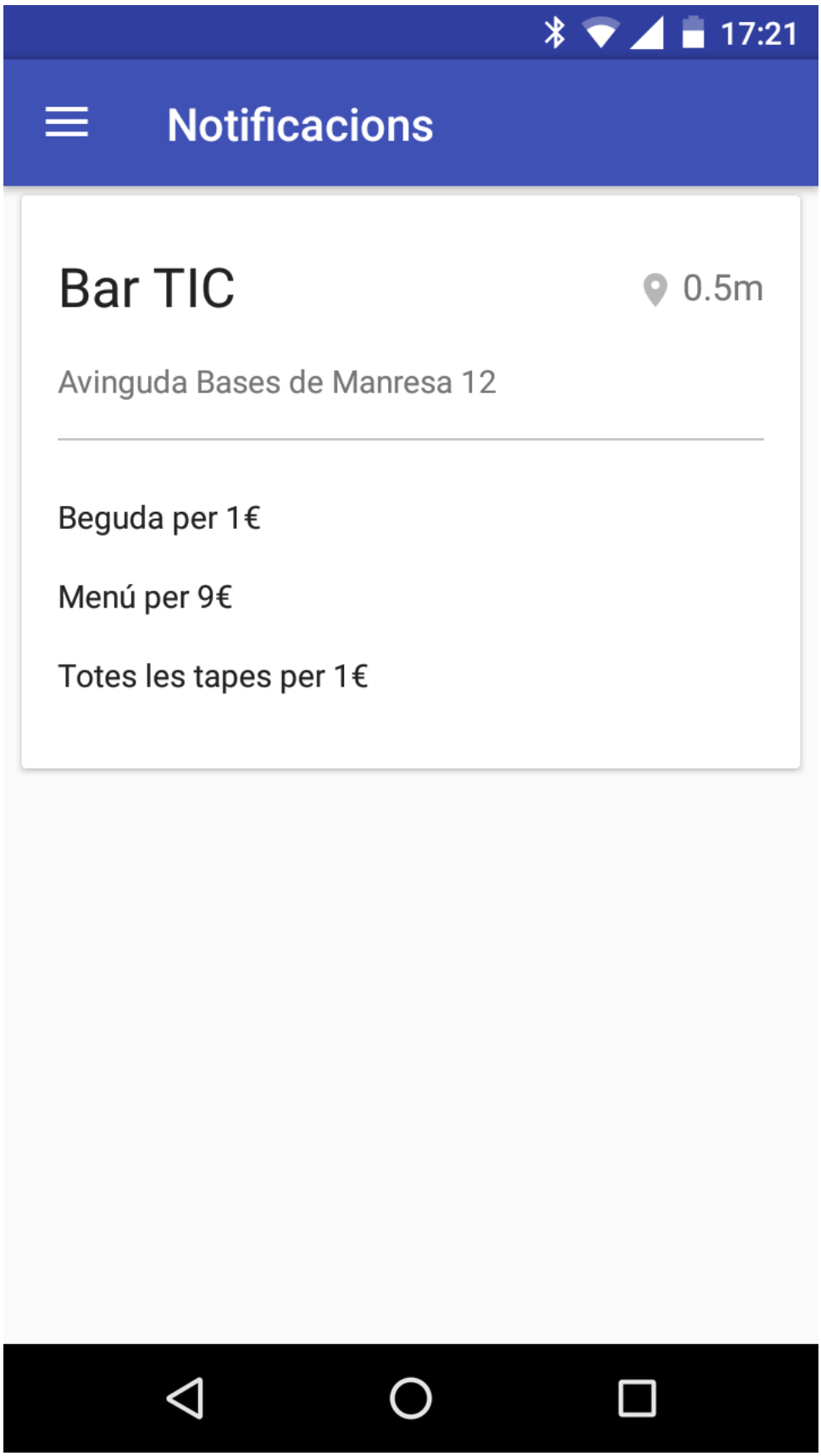

Figura 4.11.: La informació rebuda de l'aplicació veure en format de locals

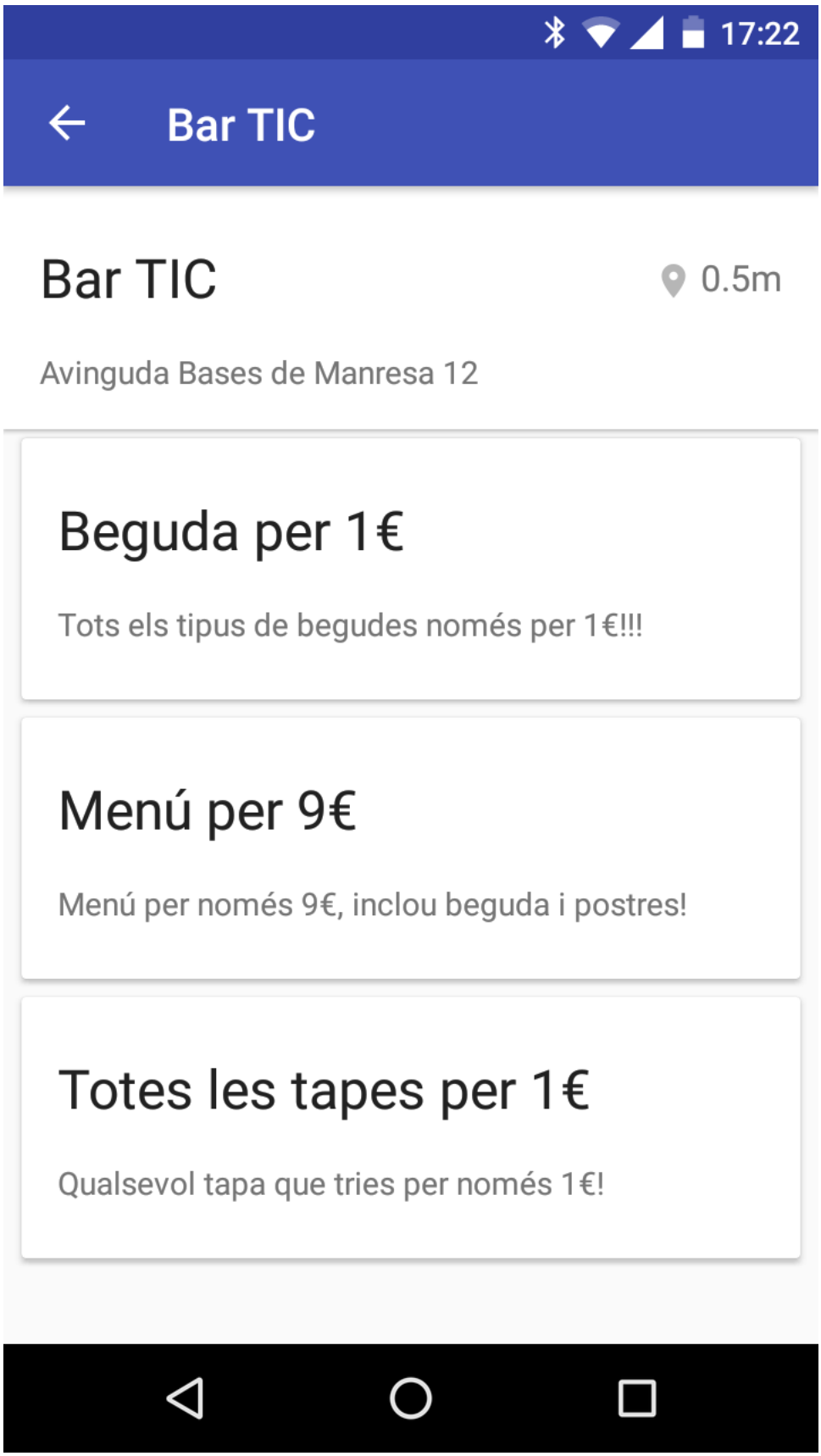

Figura 4.12.: Detalls de les notificacions d'un local

5. Conclusions

Amb aquest projecte s'ha aconseguit fer realitat un sistema de recepció de notificacions a través dels telèfons intel·ligents quan s'acosta a un punt d'interès on està situat l'emissor BLE. Per aconseguir aquest objectiu, s'ha hagut de desenvolupar un dispositiu BLE per fer d'emissor, un panell d'administració per gestionar la informació de les notificacions i una aplicació Android per fer de receptor de les notificacions.

Alhora de desenvolupar aquest projecte s'ha tingut en compte aconseguir un sistema fiable, robust i segur. Especialment en el panell d'administració web, on els diferents usuaris no poden accedir o modificar dades dels altres sense tenir accés. Durant aquest treball s'han utilitzat diferents eines, com ara, Flask, Android, Docker... i això m'ha permès conèixer més a fons coneixements prèvis que s'havien ensenyat en el grau.

Aquest projecte m'ha donat una sensació satisfactoria, ja que s'ha realitzat un treball que aplica els tres camps de l'Enginyeria de sistemes TIC.

Treballs futurs

Durant el desenvolupament del projecte, s'ha trobat que existeix unes possibles millores pels treballs futurs:

- Dissenyar una placa de circuit imprès propi per reduir el cost i la mida d'emissor BLE.
- Incorporar mecanisme de seguretat al sistema per estal·lviar consum de dades i energia dels telèfons mòbils, en el cas quan hi ha un emissor maliciós intenta fer difusions amb molts identificadors diferents.
- Implementar sessió per l'usuari d'aplicació mòbil, d'aquesta manera pot deixar que l'usuari defineixi les seves preferències de notificacions i obtingui notificacions personalitzades.

Bibliografia

[16a] *Adafruit. Introduction to Bluetooth Low Energy*. 3 d'oct. de 2016. URL: https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap.

[16b] *Android Developers. Supporting Different Platform Versions*. 12 d'oct. de 2016. URL: https://developer.android.com/training/basics/supporting-devices/platforms.html.

[16c] *Android Developers. Application Fundamentals*. 12 d'oct. de 2016. URL: https://developer.android.com/guide/components/fundamentals.html.

[16d] *Android Developers. Intents and Intent Filters*. 12 d'oct. de 2016. URL: https://developer.android.com/guide/components/intents-filters.html.

[16e] *Android Developers. Starting an Activity*. 12 d'oct. de 2016. URL: https://developer.android.com/training/basics/activity-lifecycle/starting.html.

[16f] *Android Developers. Declaring Permissions*. 12 d'oct. de 2016. URL: https://developer.android.com/training/permissions/declaring.html.

[16g] *Android Developers. BluetoothLeScanner*. 12 d'oct. de 2016. URL: https://developer.android.com/reference/android/bluetooth/le/BluetoothLeScanner.html#startScan(android.bluetooth.le.ScanCallback).

[16h] *Android Developers. Android 6.0 Changes*. 12 d'oct. de 2016. URL: https://developer.android.com/about/versions/marshmallow/android-6.0-changes.html#behavior-hardware-id.

[16i] *Android Developers. Connecting to the Network*. 12 d'oct. de 2016. URL: https://developer.android.com/training/basics/network-ops/connecting.html.

[16j] *Android Developers. Bluetooth Low Energy*. 12 d'oct. de 2016. URL: https://developer.android.com/guide/topics/connectivity/bluetooth-le.html.

[16k] *Android Developers. Layouts*. 12 d'oct. de 2016. URL: https://developer.android.com/guide/topics/ui/declaring-layout.html.

[16l] *Android Developers. Android Studio*. 12 d'oct. de 2016. URL: https://developer.android.com/studio/index.html.

[16m] *Android Devepolers. Transmitting Network Data Using Volley*. 12 d'oct. de 2016. URL: https://developer.android.com/training/volley/index.html.

[16n] *Android Dvelopers. Sending Work Requests to th Background Service*. 12 d'oct. de 2016. URL: https://developer.android.com/training/run-background-service/send-request.html.

[16o] *Arduino. Arduino Uno*. 4 d'oct. de 2016. URL: https://www.arduino.cc/en/Main/ArduinoBoardUno.

[16p] *Docker. Docker Overview*. 3 d'oct. de 2016. URL: https://docs.docker.com/engine/understanding-docker/.

[16q] *Docker. Overview of Docker Compose*. 9 d'oct. de 2016. URL: https://docs.docker.com/compose/overview/.

[16r] *Flask. Foreword - Flask Documentation (0.11)*. 3 d'oct. de 2016. URL: http://flask.pocoo.org/docs/0.11/foreword.

[16s] *Github. ble-sdk-arduino*. 1 d'oct. de 2016. URL: https://github.com/NordicSemiconductor/ble-sdk-arduino.

[16t] *Github. Eddystone Protocol Specification*. 5 d'oct. de 2016. URL: https://github.com/google/eddystone/blob/master/protocol-specification.md.

[16u] *Google Developers. Eddystone format*. 13 d'oct. de 2016. URL: https://developers.google.com/beacons/eddystone.

[16v] *In-Circuit Online Shop. Radino nRF8001, Bluetooth-Low-Energy*. 1 d'oct. de 2016. URL: http://shop.in-circuit.de/product_info.php?cPath=22_27&products_id=31.

[16w] *Material design. Material design*. 12 d'oct. de 2016. URL: https://material.google.com.

[16x] *Nordic Semiconductor. nRF8001*. 1 d'oct. de 2016. URL: http://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF8001.

[16y] *Quora. Why does GPS use so much more battery than any other antenna or sensor in a smartphone?* 1 d'oct. de 2016. URL: https://www.quora.com/Why-does-GPS-use-so-much-more-battery-than-any-other-antenna-or-sensor-in-a-smartphone.

[16z] *Viquipèdia. Sistema de posicionament global*. 1 d'oct. de 2016. URL: https://ca.wikipedia.org/wiki/Sistema_de_posicionament_global.

[16aa] *Viquipèdia. MariaDB*. 2 d'oct. de 2016. URL: https://ca.wikipedia.org/wiki/MariaDB.

[16ab] *Viquipèdia. Android*. 4 d'oct. de 2016. URL: https://ca.wikipedia.org/wiki/Android.

[16ac] *Wikipedia. Wi-Fi positioning system*. 1 d'oct. de 2016. URL: https://en.wikipedia.org/wiki/Wi-Fi_positioning_system.

[16ad] *Wikipedia. Bluetooth low energy*. 1 d'oct. de 2016. URL: https://en.wikipedia.org/wiki/Bluetooth_low_energy.

[16ae] *Wikipedia. Bluetooth*. 1 d'oct. de 2016. URL: https://en.wikipedia.org/wiki/Bluetooth.

[16af] *Wikipedia. Eddystone (Google)*. 4 d'oct. de 2016. URL: https://en.wikipedia.org/wiki/Eddystone_(Google).

[16ag] *Wikipedia. Front and back ends*. 3 d'oct. de 2016. URL: https://en.wikipedia.org/wiki/Front_and_back_ends.

[16ah] *Wikipedia. Flask (web framework)*. 3 d'oct. de 2016. URL: https://en.wikipedia.org/wiki/Flask_(web_framework).

[16ai] *Wikipedia. Web server*. 3 d'oct. de 2016. URL: https://en.wikipedia.org/wiki/Web_server.

[16aj] *Wikipedia. Continuos Delivery*. 9 d'oct. de 2016. URL: `https://en.wikipedia.org/wiki/Continuous_delivery`.

[Sem15] Nordic Semiconductor. *nRF8001 Product Specification*. Ver. 1.3. Mar. de 2015. URL: `https://www.nordicsemi.com/eng/content/download/2981/38488/file/nRF8001_PS_v1.3.pdf`.

[Tow+14] Kevin Townsend et al. *Getting Started with Bluetooth Low Energy*. First release. CA, USA: O'Reilly Media, Inc., 2014.

Part II.

Apèndixs

Nota

Durant el desenvolupament d'aquest projecte s'ha utilitzat el sistema de control de versions git. Es pot trobat codi font en el zip ajuntat o a `gitlab.jmr.cat`:

- *Aplicació Android*: `https://gitlab.jmr.cat/tfg-qiwei/client-android-app`
- *Webs*: `https://gitlab.jmr.cat/tfg-qiwei/api-web`